AFRL-IF-RS-TR-1999-156
Final Technical Report
July 1999

DISTRIBUTED SHARED WORKSPACE

BBN Technologies

Charles Blake, Maureen Doyle, David A. Karr and David Bakken

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-1999-156 has been reviewed and is approved for publication.

APPROVED: Thomas F. Lawrence

THOMAS F. LAWRENCE
Project Engineer

FOR THE DIRECTOR:

WARREN H. DEBANY, Jr., Technical Advisor
Technical Advisor
Information Grid Division

If your address has changed or if you wish to be removed from the Air Force Research Laboratory Rome Research Site mailing list, or if the addressee is no longer employed by your organization, please notify AFRL/IFGA, 525 Brooks Road, Rome, NY 13441-4505. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	Jul 99	Final Apr 96 - Jun 98

4. TITLE AND SUBTITLE

DISTRIBUTED SHARED WORKSPACE

5. FUNDING NUMBERS

C - F30602-96-C-0008
PE - 62702F
PR - 5581
TA - 21
WU - AL

6. AUTHOR(S)

Charles Blake, Maureen Doyle, David A. Karr, and David E. Bakken

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

BBN Technologies
10 Moulton Street
Cambridge, MA 02138

8. PERFORMING ORGANIZATION REPORT NUMBER

N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

AFRL/IFGA
525 Brooks Road
Rome, NY 13441-4505

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

AFRL-IF-RS-TR-1999-156

11. SUPPLEMENTARY NOTES

AFRL Project Engineer: Thomas F. Lawrence, IFGA, 315-330-2925

12a. DISTRIBUTION AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

N/A

13. ABSTRACT *(Maximum 200 words)*

The objective of the DSW project is to investigate the concept of a collaborative workspace. This workspace should allow large communities of users, working at long distances, to collaborate as effectively as if they are in a single room talking face to face and have access to information regardless of its native location around the world. The objective of DSW is closely related to the field of computer-supported cooperative work. While most efforts in that field have concentrated on user interfaces and data structures, relatively little attention has been paid to whether systems incorporating the new features scale well as parts of the system are distributed over various distances, including local-area networks and wide-area networks. DSW addresses this concern. The results of this project include the following: 1) Identification of special issues in the design and usage of distributed collaborative applications, including the problems of consistency and confidence in the data and decisions managed by such applications, as well as the need to adapt to the varying performance capabilities of the underlying computing resources; 2) Identification of a spectrum of notions for distributed consistency that are applicable to distributed collaborative applications; and 3) Compilation of a set of applicable metrics to describe quality of service in distributed collaborative applications.

14. SUBJECT TERMS

Distributed Shared Workspace, Collaborative Workspace, Quality of Service (QoS), Mutual Consistency, Adaptivity

15. NUMBER OF PAGES

40

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Contents

1 Project Summary

The objective of the DSW project is to investigate the concept of a collaborative workspace. This workspace should allow large communities of users, working at long distances, to collaborate as effectively as if they are in a single room talking face to face and have access to information regardless of its native location around the world.

The objective of DSW is closely related to the field of computer-supported cooperative work. While most efforts in that field have concentrated on user interfaces and data structures, relatively little attention has been paid to whether systems incorporating the new features scale well as parts of the system are distributed over various distances, including local-area networks (LANs) and wide-area networks (WANs). DSW addresses this concern.

The results of this project include the following:

1. We identified special issues in the design and usage of distributed collaborative applications, including the problems of consistency of and confidence in the data and decisions managed by such applications, as well as the need to adapt to the varying performance capabilities of the underlying computing resources.

2. We identified a spectrum of notions of distributed consistency that are applicable to distributed collaborative applications.

3. We compiled a set of applicable metrics to describe quality of service in distributed collaborative applications.

4. We developed a hypothetical collaborative application for the purpose of illustrating the principles of DSW. (In addition, an actual demonstration application is being presented as a separate deliverable.)

5. We applied the principles of DSW to a discussion of a distributed collaborative application (the OpenMap project) that is being developed in another project at BBN Technologies. This discussion includes detailed alternative algorithms for adaptation.

6. We collected experimental statistics on an adaptive CORBA application, showing how adaptation is affected by system performance characteristics, in this case the characteristics of the underlying network.

7. We submitted the results and lessons learned to the *19th International Conference on Distributed Computing Systems (ICDCS'99)*; see [REF-06].

A list of references used in this work appears in an appendix.

2 Glossary

CORBA Common Object Request Broker Architecture. The OMG's distributed object standard; see OMG below.

DSW Distributed Shared Workspace. This project.

LAN Local-Area Network.

OMG Object Management Group. The largest consortium in the world, with over 500 software and hardware vendors, end users, and government agencies. The OMG is developing the CORBA standard. See URL http://www.omg.org.

QoS Quality of Service.

QuO Quality Objects, a quality of service framework for CORBA Objects.

WAN Wide-Area Network.

3 Technical Overview

In order to understand the concept of a collaboration workspace, in this section we explain the background of working collaboratively, and the particular area of collaboration that DSW addresses. Section 3.1 defines the broad area of worldwide information systems and how it ties to a distributed shared workspace vision, and explains how simultaneous collaboration is required to satisfy the performance requirements for collaboration in heterogeneous environments. Section 3.2 summarizes the essential collaboration problem that is of interest here. Section 3.3 presents a detailed analysis of a hypothetical crisis management application to illustrate the objectives that guided the DSW project, including the demonstration to be delivered separately from this report. Finally, section 3.4 defines a spectrum of consistency criteria that we found to apply to this area of research.

3.1 Introduction

3.1.1 Worldwide Information Systems and DSW Collaboration Vision

Wide-area, interoperable access to information, services and more complex facilities is becoming available through a variety of technologies. Support to coordinate the activities of widely distributed users is being supported through workflow management and similar approaches. And attempts to support collaboration among simultaneous users is illustrated by game software and by efforts to juxtapose virtual environments with video-conferencing along with planning tools and shared information spaces.

These approaches serve when network delays are modest, availability is high and workspace semantics are simple and supported by a few vendor applications. Scalability to large groups of users is not well supported except perhaps for the use of internet multicast support for audio and video streams; these techniques do not translate well to the actual data elements that occupy a shared internet workspace. The existing architectures do not provide abstractions that work well to hide choices of underlying prototype types such as point-to-point versus multicast, nor deal with the adaptations that might help an end-user or agent negotiate how available communication should be applied to complex data transfers to meet specific requirements. Examples might include reducing video quality to allow more rapid data transfer, or reducing the level of detail or the amount of backup material that should be pre-fetched to allow for more timely delivery of complete overview material.

Through our initial effort in this area, we are interested in pursuing models that are appropriate for simultaneous human-human and human-agent collaboration, where data consistency can be relaxed to adapt to varying availability so long as the people and agents using the system can still make confident, correct decisions based on the data presented to them. Our focus is on creating a shared object environment that can capture the essential "transaction" requirements of an application, and validating this approach through use in building applications. This architecture can, over time, allow choices of which objects and data elements are transferred and when those updates occur to be handled by policy algorithms that can be written relatively independently of the applications. It will

also allow choices of underlying transport type to be encapsulated, with the reliability strategy chosen to match the transport type and data quality requirements of particular users. We are, for the moment, focused on implementing these mechanisms in the visual and event space of human users; however, the algorithms and techniques we develop are likely to be suitable for replicating persistent storage in object databases, in CORBA services and in Java, and other means used to deliver distributed information services.

3.1.2 Simultaneous Collaboration: Consistency, Confidence, Adaptation

The growth of web based information systems and the growing use of distributed object systems such as CORBA are creating a foundation for a common information workspace that can be used for collaboration. The technical infrastructure—web servers, object request brokers, the Java language and environment, and so forth—create a distributed computing context that facilitates access to remote information systems and services. It also facilitates integration efforts through user interfaces, mediators, and agents that combine data from different sources. Advanced development efforts, such as the DARPA ISO funded Joint Task Force Advanced Technology Demonstration (JTF-ATD), are specifically intended to exploit this potential through the integration of new and existing information accessed through servers and available for accelerated application development.

Our goal is to build one aspect of this infrastructure to the point of a true "Distributed Shared Workspace (DSW)," that is scaleable to use by a large number of varying simultaneous users and can be made to operate effectively in a wide area environment. Building such a workspace requires that we address data consistency and event synchronization in ways to help users sustain a high level of confidence in the collaborative decisions they make, while the system adapts to the varying communication efficiency of internet protocols, congestion and availability. The system may also need to make use of active resource management mechanisms to promote efficient progress of important, competing activities that share the resources of the workspace.

We are also setting out to develop the shared object model as a foundation for granting greater adaptivity to information management in distributed information systems. What we propose is not in conflict with the distributed object client/server model, but rather creates a significantly stronger model that will reduce the need for the developers to deal directly with complex mechanisms, such as proxy objects, that current distributed object computing environments require for performance management. Object sharing can allow the developer to focus initially on an object's functional implementation, defining its essential methods. Object sharing then allows additional code to be added that captures policy algorithms. Although the functional code may need to be altered to more carefully capture how changes must be clustered into transactions or other aggregate groupings, the policy code controls how and when the exchange of these changes is scheduled, dynamically, as communication availability changes. The schedule may also be based on a forecast of future availability. Policy also includes the potential to vary the accuracy or level of detail of the object that is transmitted. Finally, this framework allows for the potential that the policy algorithm may treat members of a conference differently, perhaps reducing the accuracy or detail of information sent to users on low-speed links to allow them to keep up with the pace of the conference, which allowing them the opportunity to improve the accuracy or detail on demand when needed. These goals go

beyond those of Java, which is an effective tool for hiding whether methods are implemented locally or remotely; object sharing includes consideration of persistence and consistency as steps toward a framework that allows the policy algorithms to be treated separately from functional code, which in turn is a step toward greater adaptivity.

3.2 The Collaboration Problem

The strictest version of simultaneous collaboration would incur significant delays since each program would move in lockstep, committing changes with all parties before moving to the next step. We are looking for a weaker, more highly available form of consistency, where some effects may be delayed to cope with network congestion, but still appear in a coherent form when the data are finally available. We are also looking to build, in the longer term, a combination of middleware, high level abstractions, along with design patterns and programming techniques that will simplify the task of developing these applications.

3.3 Crisis Response as a Problem-Focusing Context

The scope of DSW includes the definition and development of a distributed collaborative planning scenario. This section describes distributed collaborative planning for a crisis response, and the properties of the networks during crisis response.

3.3.1 Crisis Response Definition

In a crisis, high-level commanders are assigned. These commanders choose subordinates from various branches of the military based on selections available or mandated from their higher commanders. The process iterates until it reaches fixed non-crisis command structures. At each level the commander issues a high-level situation description and desired end-states and the subordinate commander devises more specific plans to accomplish mission goals. The superior commander selects from proffered alternative plans, resolves resource allocation issues, and approves going forward. The planning process is often iterative and generally undergoes revision all during its execution.

In the initial plan development phases, and during subsequent re-planning, subordinate commanders communicate with their superiors and their own subordinate commanders and other support infrastructure providing information relevant to all command levels, such as meteorological and logistical data.

In a crisis, timeliness is a critical aspect of planning. It is often unrealistic to gather together all the parties who need to participate and impossible to maintain such physical collocation. The participants need to collaborate from widely dispersed locations as needed. This is even more necessary in the deployment and execution phases of an operation.

While, there are of course minor exceptions to this general framework, this description captures the flavor of many military planning doctrines. A main goal of the DSW project is to support providing better computer-supported tools for enhancing the collaborative planning and re-planning process.

3.3.2 Properties of a Network in Crisis

Communications support and requirements vary widely for command structures in a military response to a crisis. Commanders and all other soldiers are initially assigned to well-established military infrastructures, typically full military bases of operations or other kinds of battle groups.

As execution of a plan unfolds, component forces rapidly deploy outside this established infrastructure. For instance, a CJTF is often deployed in a matter of days. Planning, of necessity, continues throughout deployment and execution, perhaps with poor or no communications with the experts in garrison. Networks are built, configured, and re-configured. Yet commanders have critical communications requirements for returning information about the unfolding situation and mission status, for receiving up to date data from specialized sources, and for re-planning in conjunction with superior and inferior commanders and the changing landscape of the problems being solved.

Particularly in a crisis, brand new military network infrastructures are often deployed with time of the utmost essence. Besides time constraints, other constraints imposed by the particular crisis at hand can make these networks far from optimal as a basis for collaborative planning.

3.3.2.1 Intermittent Connectivity

The dynamic nature and restructuring of the network often introduces intermittent connectivity. Various competing demands on the network often saturate the available bandwidth making effective latencies high due to packet congestion. Discovery of new routes when subsets of network nodes go off-line due to power system reconfiguration or outages can also wreck havoc with connectivity.

3.3.2.2 High Latency

Even without such congestion, the widely dispersed nature of United States military operations, as well as many commercial endeavors does not lend itself to dramatic revision. The sheer distance scales involved and fundamental physical limitations have significant implications for cross-continental communication round-trip latencies.

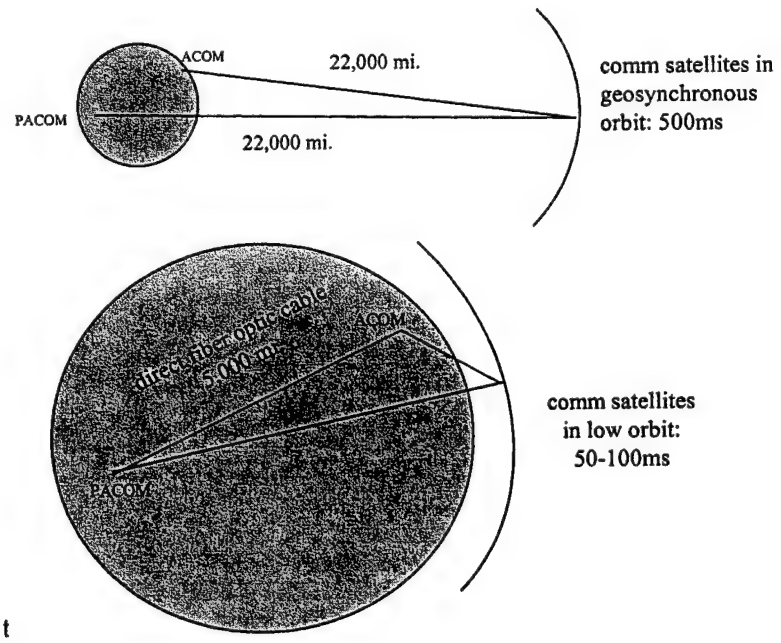
Current telecommunication infrastructure includes thousands of satellites in geosynchronous orbits 22,000 miles above the surface of the Earth (see Figure 1). The speed of light mandates 236 milliseconds one-way transit delay for a packet to travel from a ground-based relay to the satellite and back. Round-trip delays are typically nearly 500 milliseconds. Global communications routes may even require multiple ground-space hops.

These times rapidly add up to network latencies unacceptable for simultaneous collaboration with traditional client/server methodologies. While installation efforts are underway for both land-based fiber-optic lines and low-flying satellite arrays, the actually physical space traversed by the light is still likely to be a significant multiple of the minimum physical distance. Additionally, the military environments DSW aims to support require network connections between highly mobile units, both on land and at sea. Such mobile units are likely to rely on satellite signal relay for the foreseeable future.

Figure 1 illustrates some of the scales involved. The great circle distance in the diagram may be somewhat misleading since the absolute minimum distance between Norfolk and Hawaii is 5,000 miles, but it is very unlikely that this trajectory will be realizable in practice. More tangential routes on the order of twice the minimum distance (or more) are the likely outcome of low-flying satellite arrays. These sorts of paths will yield round-trip times on the order of 100 ms. While a substantial improvement over the present situation, this is still only marginally better than the 160 ms typical with 28.8 KBPS analog modems. Round-trip times of this order have been shown to make applications designed for low latency, such as remote X Windows display protocols, intolerably sluggish.

3.3.2.3 Routing Problems and Partitioned Networks

In addition to bandwidth, latency, and intermittent connectivity, crisis networks are also characterized by routing difficulties. A partitioned network is one where host A can exchange packets with host B, and host B can exchange packets with host C, but, unfortunately, host A cannot exchange packets with



• Figure 1: Global Area Latency: Current and Future

host C. Ordinarily dynamic routing algorithms are designed to prevent this situation. Rapid initial configuration, frequent reconfiguration, and mobile units within crisis networks contribute to make partitioned networks. Static routes as default policies in many parts of the non-crisis network, which the crisis network relies on also contribute to partitioned networks. The default static route policies are often based on issues involving administrative control over which packets are actually routed, rather than technical difficulties. A consequence of this foiling of dynamic routing algorithms is that network partitioning is commonplace.

The consequence for the end user of network partitioning depends greatly on the nature of the application. Because this situation is assumed to either be handled by the routers or be insoluble, almost no programs and protocols either detect or adapt to it. If the application is a simple text-oriented program, the user might be able to adapt via successive logins. But if the application is a collaborative application or a more complex network-oriented client-server application it is almost certain that full A-B-C collaboration will be impossible. The reason will often be due to administration difficulties and not to authorization issues. This problem, however, is solvable if the distributed applications in question detect the situation and adopt message-forwarding policies in lieu of the routers. Such behavior could be optional and security policies enforced either by disabling or forcing users to be authorized before activation of the internal routing compensation.

3.3.2.4 Bandwidth Variance over Times and Hosts

Beyond the functioning of the network, crisis networks are also characterized by extreme variance of the network performance in bandwidth, latency, and reliability. The variance is both across hosts in the network and over the course of time. Clearly functioning in and adapting to such a dramatically variable and fundamentally limited communications substrate is difficult. This is the primary constraint the crisis context imposes upon the design and implementation of DSW. Additional information from quality of service facilities can further improve the detection and response to highly variable networks.

3.3.2.5 Open-Loop Competition for Resources

In addition to poor baseline network characteristics, communication resources are in high demand by many parties for many reasons. In a crisis the demand is even greater. The result is that the limited network resources available tend to become and remain stretched to the limits of their capacities. This open-loop competition between users of the network typically occurs across a large set of network protocols. Besides DSW protocols, this set may include large numbers of e-mail transfers, HTTP web page transfers, file transfers, audio and video teleconferencing, heavyweight ORB protocols, data base queries and other more specialized traffic. Any particular user or application may only be able to utilize a small slice of network transmission capability. Finite memory buffers on either side of the links may overflow causing many re-transmissions to occur. The re-transmissions increase the effective traffic, compounding the problem. The end result is that networks under heavy loads suffer from extremely degraded performance.

Land-sea communications links are a contemporary example of how degraded performance can become in these circumstances. These links may realistically consist of a single 14.4 Kbps secure STU III modem link shared between all applications and protocols on the entire vessel. Effective round-trip latencies can soar to over 30 seconds and bandwidth can plummet to mere dozens of bytes per second. The situation can be so poor that many protocols can simply mistake poor performance for a disconnected line. Even typing in a dedicated terminal application can become intolerably slow.

The same factors which make for a poorly optimized communication infrastructure are often the very same factors which make collaboration so critical.

Having examined the kinds of problems we need to solve and the environmental constraints in which we desire to operate within, we now motivate a proposed system design from a bottom up analysis, mentioning top-down constraints as relevant.

3.4 A Spectrum of Notions of Distributed Consistency

Distributed consistency is a term meant to refer to how multiple parties might *agree* on the values of some shared data. There are a number of different "agreement models" which we might target. Since distributed consistency maintenance has such a central role to the implementation of the toolkit, it is appropriate to discuss some of the options. The options group into two categories. The term *access* will often be used to mean *any* access to shared data, either reading or writing. We shall describe these two categories, give examples in table form, and then motivate the choice we have decided is most appropriate for the simultaneous collaboration portion of a shared work environment.

Techniques of the first category, models with explicit synchronization, achieve consistency through by explicit waiting. In this type of model, any process which needs to modify the distributed data needs to acquire permission to do so from the group of processes. Once this permission is had, the update

• Table 1: Consistency Models with Explicit Synchronization

Consistency	Description
Weak	Accesses to locks are sequentially consistent. No access to a lock is allowed until all previous writes have completed everywhere. No access to objects is allowed until all pending lock accesses are complete.
Release	Before an access to an object is performed, all previous lock acquisitions must have completed successfully. Before a lock is released, all previous read and writes done by the process must have completed propagating. Acquire and release accesses to locks must be PRAM consistent (see Table 2).
Lazy release	Like release consistency, but sending all the update data is deferred until some process acquires a lock.

• Table 2: Consistency Models without Explicit Synchronization

Consistency	Description
Strict	Any read of an object returns the value stored by the most recent write operation to the object. This is the single-user consistency model.
Sequential	The result of any execution is the same as if the access of all processors were executed in some sequential order, and the accesses of each individual processor appear in this sequence in the order specified by its program
Causal	Accesses that are potentially causally related must be seen by all processes in the same order. Concurrent writes may be seen in a different order on different machines.
Pipelined RAM (PRAM)	Writes done by a single process are received by all other processes in the order in which they were issued, but writes from different processes may be seen in a different order by different processes.

can be done and the permission released. Permission to modify data is often referred to as a "lock". Table 1 has a brief summary of some of the more commonly used forms of explicit synchronization models.

A general feature of all these models is that before any access to the shared data is done, the entire group of participants has to grant access. This is a message-heavy exchange, which does not fare well in a high latency environment or an environment where there are many participants over a WAN.

It is generally the case in distributed systems that more messaging is required to ensure stronger guarantees. Each successive entry in Table 1 requires less messaging overhead, but gives fewer guarantees.

Techniques of the second category, models without explicit synchronization, are given in Table 2. These techniques achieve consistency by enforcing some order to the visibility of updates. In this type of model, a process may need to wait for read or write access to data for some unspecified period of time, but the read or write may be implicitly delayed. No explicit access token, like a lock, is required.

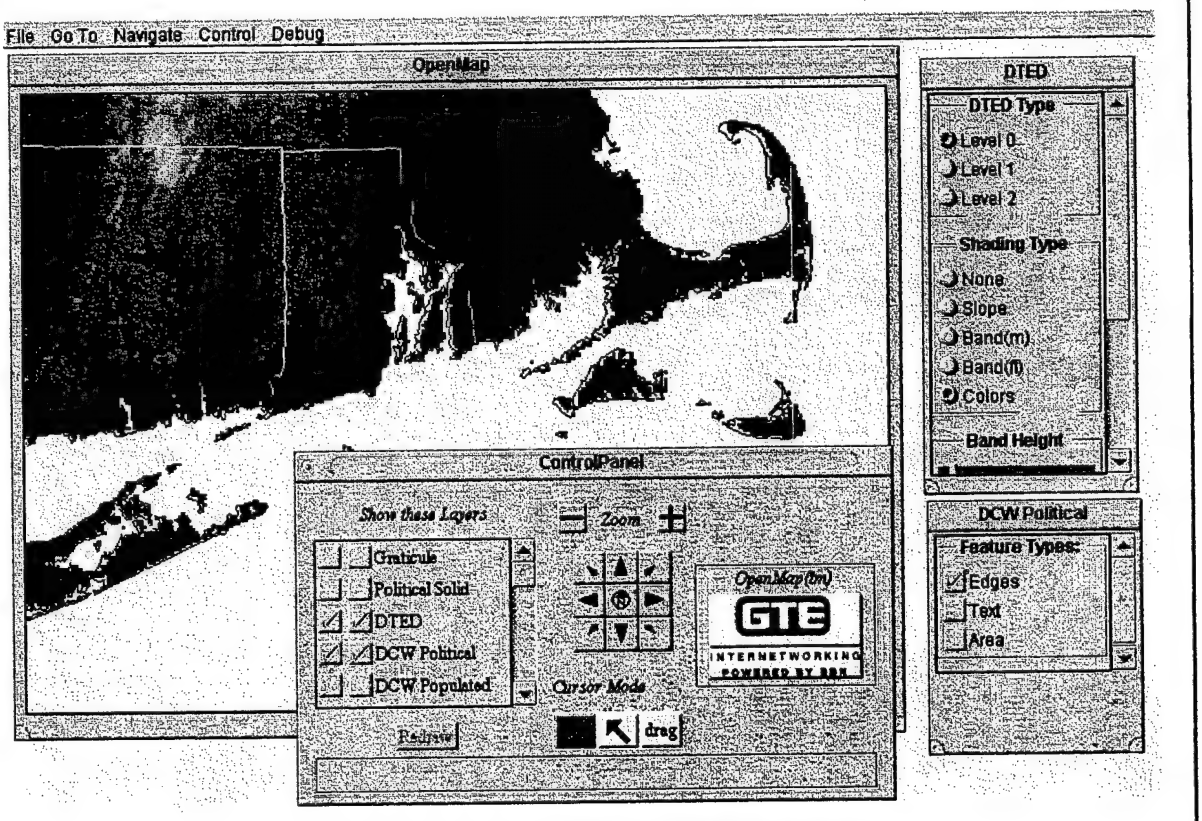
Agreement on the values of data is either a result of expensive group updates or a result of "convergence" after some finite period of time. This latter process is of particular concern for DSW. When the writing stops, the values of the various copies of the data will converge. In the PRAM model with a timestamp protocol, when writes stop happening, the values will stabilize to the last written value. All parties will agree upon this final ordering. The exact time it takes to reach this final agreement will be a complex function of the number of communicating parties and communication latency between each party.

4 Collaborative Workspaces and Quality of Service (QoS)

This section concretely relates quality of service to issues in distributed collaborative systems. In Section 4.1 we describe experiences with distributed collaborative applications at BBN Technologies that illustrate the need for adaptation and control over quality of service (QoS). In section 4.2 we identify and address the particular requirements that arise from these experiences.

4.1 The Need for QoS in DSW: Experiences from MATT

BBN has been developing distributed middleware, and applications using it, for more than two decades. In particular, we have been developing collaborative mapping software for the US Department of Defense since 1987. The issues we address in this report, and our proposed approach to addressing them, arise out of these experiences. A good example of the experiences developers face due to the lack of QoS metrics and support is seen in the development of the Mapping Analysis Tool for Transportation (MATT).



• Figure 2 : OpenMap™ Environment

BBN completed development of MATT for USTRANSCOM in 1994 [REF-03]. MATT enables transportation analysts to browse world maps and request geographically referenced information. The MATT software forms the basis for BBN's current work on the JTF Map System, which provides distributed collaborative planning using the latest object-oriented programming concepts and three-tiered client-server technology. This software provides mapping services for the JTF Planner, Advanced Mobility Platform, Logistics Anchor Desk [REF-04], Voice-Activated Logistics Anchor Desk [REF-05], METOC Anchor Desk [REF-01], and TRAC2ES. These systems have been fielded in a wide variety of environments and locations, including Desert Storm and Bosnia.

MATT's architecture supports the incorporation of many different kinds of geographic data, derived from distributed data sources. Each data type is accessed by a CORBA data server called a *specialist.* Specialists provide the translation from semantically rich geographic information sources to basic geographically referenced objects to be rendered. A map viewer application presents the objects from diverse data sources in multiple layers overlaid on a single display, which provides a shared map workspace for collaboration. A user's view may change due to changes in the data underlying the layers — for example, an object's status could change and the icon representing it must be similarly changed — or due to interactive annotations by other users. Applications can incorporate the MATT viewer capability into their particular GUI, while using specialists to add domain-specific collaboration functionality. Example domains have included transportation and logistics planning. OpenMap™ is a next-generation implementation of the MATT architecture, built in Java and based on emerging open standards for geospatial information exchange [REF-02]. A representative example of a MATT or OpenMap session is given in Figure 2.

4.1.1 The Choice-of-Implementations Dilemma

MATT allows a user to select a geographic region on a map in the workspace, and to request the application to render all the applicable objects in that region. We now explain what steps were involved in doing this, why some algorithms worked well in some conditions and poorly in others, and how systemic limitations (in particular, the lack of QoS support) impeded an experienced programmer from producing a generally acceptable implementation.

```
render_region(geoLayer, region) {
      obj_ref_list = geoLayer.get_objects(region);
      foreach geoObject in obj_ref_list
            render_geo_object(geoObject);
}

render_geo_object(geoObject) {
      ...
      latitude = geoObject.getLatitude();
      longitude = geoObject.getLongitude();
      glyph = geoObject.getGlyph();
      ...
      render(latitude, longitude, glyph, ...);
}
```

• Figure 3: Pseudocode for Natural Object Access Implementation

4.1.1.1 Implementation 1: Natural Object Access

Initially, the programmer responsible for selection and rendering of a region implemented it in a fashion represented by the pseudocode given in Figure 3.

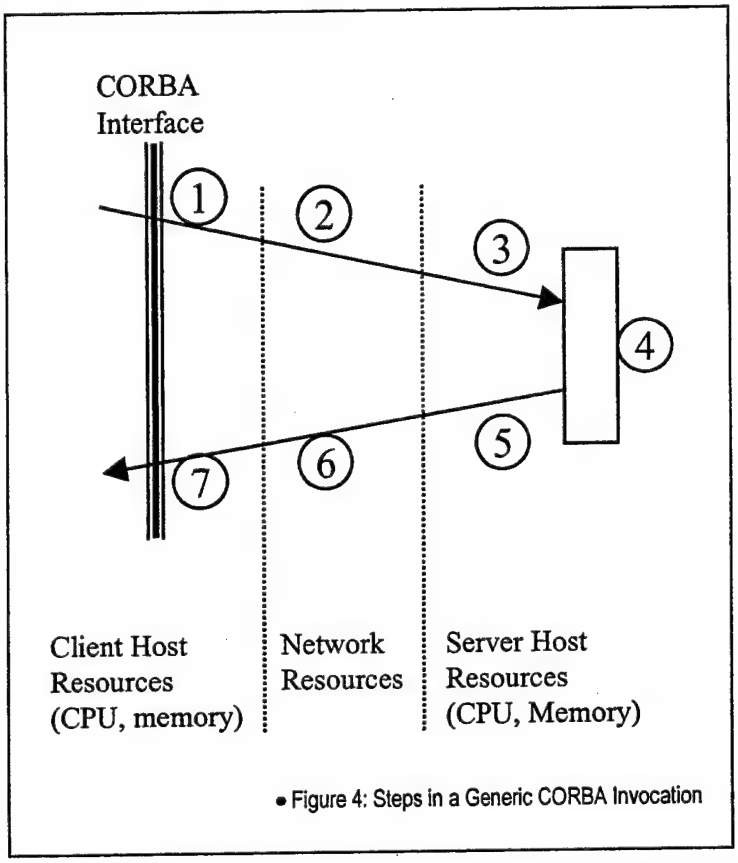
Each layer of the map is displayed by making a call to `render_region()`, which renders all objects that fall within that region. The first step obtains a list of references to these objects, often numbering in the thousands. The second step iterates over the list and renders each object (represented by the variable `geoObject`). To render `geoObject`, several attributes of the object are needed to determine the representation and location of the object. Each attribute is obtained by a separate method call on `geoObject`.

This implementation accesses objects in the manner that the programmer considered most natural. Since this code was developed initially to run on a single host, it did not interpose an ORB between the client and the server objects. Under this set of circumstances, the implementation worked well.

In the distributed implementation of the map, however, the client and the servers for the `geoObjects` ran on separate hosts, and the calls to `geoObject` became remote method invocations using CORBA. The basic steps for a generic CORBA remote method invocation are as follows:

1. Marshal the request message
2. Transmit the request message to the server host
3. Unmarshal the request message
4. Execute the server's method
5. Marshal the reply message
6. Transmit the reply message to the client's host
7. Unmarshal the reply message.

Figure 4 illustrates these steps and the physical resources they use.



• Figure 4: Steps in a Generic CORBA Invocation

As discussed previously, this implementation performed poorly when an ORB was interposed between the client and the server object. This was because the marshaling costs (Steps 1, 3, 5, and 7) were many times more expensive than a local method call. This was often a prohibitive cost, since the methods to get `geoObject` attributes would be called thousands of times. The implementation performed even worse when the client and server were placed over a WAN; in this case the communication costs (Steps 2 and 6) dominated.

4.1.1.2 *Implementation 2: Aggressive Pre-Fetching and Caching*

The second implementation of `render_region()` attempted to work around the limitations of the first by aggressively traversing all the objects to be retrieved, prefetching the attributes of each object that were most likely to be needed by the client, and bundling all this information into one large data structure. All this was performed on the server and the result passed back in one large structure. This structure included references to the remote objects, in case less-frequently-requested attributes needed to be queried. Basically, the second implementation changed the data type of the return value of `get_objects()` from a list of object pointers to a list of structures. Thus, most of the accesses to `geoObject` were local, but at the cost of increasing the size and complexity of `get_objects()`'s return value.

This implementation delivered near-optimal performance in terms overall latency for `render_region()` in the WAN case, because one large message was sent rather than thousands of small messages achieving much higher overall throughput. But this implementation would be considered inefficient if the map application were implemented in the same address space, because it copies all the `geoObject`'s states.

On a LAN, however, it performed poorly in comparison to other possible implementations. Because the attributes of all objects were returned in one large call, there was no opportunity to pipeline the process. Pipelining could include rendering `geoObjects` while others are still being transmitted or even pipelining the marshaling, transmission, and unmarshaling of the `geoObject` list itself (steps 5, 6, and 7). Because the cost of marshaling the large call (steps 5 and 7) are comparable to the cost of rendering and these operation use different resources (client CPU and server CPU) the overall latency of `render_region()` operation could have been accelerated significantly by performing those two subtasks in parallel.

Another reason to pipeline is to give early feed back to the user, that is, to display geographical data as they come in. For the WAN, pipelining does not benefit the overall latency of `render_region()`, because the transmission time for sending the results of `get_objects()` (Step 6) dominates the other communication steps and the rendering time. Pipelining may even make the latency worse because to implement pipelining involves adding new message headers which will increase the amount of data that must be sent as part of the return value, hence increasing the transmission time. Yet pipelining may be still be desirable for early feedback reasons. Hence this dilemma can only be resolved with additional information from the users of the system. For example, web browsers have a preference option to allow images to be displayed in progressive stages while they are being received.

4.1.1.3 *Implementation 3: Streaming Messages*

An alternative implementation is to use socket-based implementation of `get_object()`, where the client would send a request message and continuously poll for a sequence of messages containing the geospatial object to be rendered. This technique is often called streaming and is much in favor in some research communities. It allows for the fullest pipeline parallelism possible. This

implementation was rejected prior to coding, however, because it would have abandoned the object-oriented interfaces; these are considered to provide many software engineering benefits over the code's life cycle. It also would have required too much rewriting of the code. We are currently investigating using a form of the CORBA event service to stream `geoObjects` between the client and the server. But even this scheme will involve a drastic restructuring of the basic `render_region()`.

4.1.1.4 Implementation 4: Decomposition Layer

Another alternative implementation does not change the structure of `render_region()`, but still implements pipelining of for the `geoObject` list. This implementation places a thin layer above the CORBA interface for `get_objects()` which is transparent to the rest of `render_region()`. This layer decomposes the geographical region into a number of subregions, and invokes `get_objects()` on each subregion in parallel, collects all the replies, and combines them into one reply to the client.

This allows for a good deal of pipeline parallelism with as few as four or nine subregions. It also involves no change to the API that the server object presents to the client, merely the relatively simple insertion of a "plug compatible" layer. But it still requires a great deal of experimenting with the scheme of decomposing the main region to find a scheme that will perform adequately in all environments in which the program has to be deployed. But note that for reason mention above, this implementation would improve latency on the LAN, but would actually degrade some performance parameters on the WAN.

A variation of this implementation is to specify the same large data set, but to request only a specific part of the data to be returned, e.g., only the first half of the data, or only the third tenth. The entire data set can be retrieved by an appropriate sequence of calls that retrieve all the pieces, while enjoying all the parallelism of the decomposition layer. Such an approach, however, requires a modification of the interface and so is not plug-compatible.

4.1.2 Lessons Learned

There are a number of important lessons to be learned from the implementation possibilities of this one function. In all cases, three things indicate which algorithm to use:

1. Resource capacity (client CPU, server CPU, communication links, etc.)
2. Usage patterns (number of calls, size of the calls, etc.)
3. QoS requirements (low latency, early feedback, etc.)

These three items cannot be ascertained until runtime. For example, the application can be fielded on different hardware and network configuration (resource capacity); the user can choose different geographic regions of dramatically different sizes (usage pattern); and the user can change his or her desire for early feedback (QoS requirements).

These implementations are very different, and thus without any systematic support to capture resource availability, usage patterns, and choose the best algorithm, porting the method to perform over a different environment requires a major rewrite of the code. Worse, this code now no longer works very well, if at all, in the old environment!

Thus, the implementation of just this one function would have benefited greatly from a framework which could:

1. Allow the development of different implementations of the same functional interface.
2. Gather the information about resource contention, etc. for the application.
3. Provide a way to specify the usage patterns for the client.
4. Automatically deploy the best implementation of a module for the current resource conditions and usage patterns.

Note that the above discussions did not involve the semantics of geospatial data, just generic interactions, so any such framework would have potential for reuse across many kinds of collaborative workspace applications. The QuO framework was designed to do precisely the above. Furthermore, QuO lets you mix these QoS design patterns with any interface (specified in IDL), permitting huge labor savings and performance which will often be close to hand-coded implementations.

4.2 QoS Requirements for Collaborative Workspace Applications

Quality of service (QoS) is an established term in the multimedia application domain, frequently referring to the end-to-end performance characteristics of the delivery of a unidirectional video stream. However, video is only one possible component of an application. We thus wish to upgrade this limited understanding of QoS to better suit more general distributed application needs, especially those from the area of collaborative workspaces.

A more complete picture of a typical collaborative application (based loosely on MATT) is given in Figure 5. It shows the multiple levels of interactions between just two of the users collaborating to develop a joint logistical or transportation plan. We will discuss these levels to help illustrate how a QoS system should be able to accommodate not just a narrow definition of QoS, but should be useful to the more diverse range of typical collaborative workspace requirements.

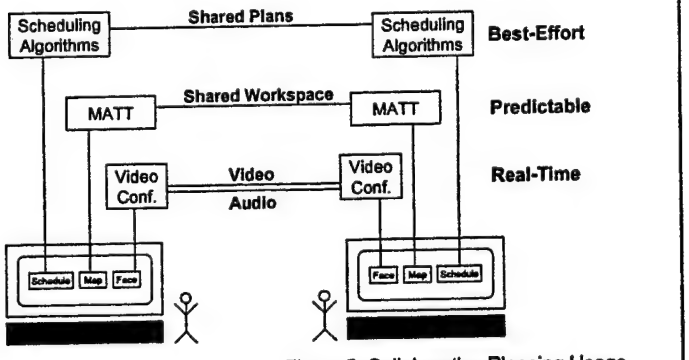
The bottom level of interaction involves a videoconference between the participants. The middle level consists of the geographical shared workspace that MATT provides. The top level consists of shared plans that are created iteratively by complex scheduling algorithms based on user inputs. These levels of interaction are quite different with respect not only to the kinds of reservations that would be required to provide the desired QoS, but also in the ways in which they can and should adapt when there are insufficient resources to provide the desired QoS at all levels.

The video and audio streams require real-time reservations to provide the timeliness (in this case reasonably low latency and jitter) required for them to be useful. Such reservations are feasible because the resource requirements for such video streams are well known. The shared workspace provided by MATT must have enough resources so the updates are predictable, or its utility to the users will rapidly decline. The scheduling algorithms do not need to be predictable, because they can take hours to run and hence the users are not waiting in real time for them to complete. Indeed, providing any sort of predictability for this layer would be infeasible; the number and duration of its interactions cannot practically be computed in advance, and we do not know what resources to set aside for them.

These layers also differ in the ways in which they can adapt to degraded resources and provide the best service possible given the existing conditions. In the videoconference layer, the video stream has ample opportunity to run at a lower update rate and still be reasonably useful to the user. The audio stream, however, can *not* be scaled back much less and still be useful. The MATT layer providing a shared workspace can be scaled back from providing 5-second updates to 30-second updates, for example. At some point, however, it rapidly becomes useless to the users if the upgrades degrade any further. Finally, the shared plan layer has great flexibility in adapting to fewer resources because no user is waiting in real time for it.

Thus, in an application such as this, a shared workspace, and an audio component, the video component may offer value to a certain set of users, or may offer little value to other users or to the same user with different collaborators or in a different environment.

QoS involves the delivery of whatever service the user needs (such as information or interaction with a coworker): how *timely* this delivery is, how *much* is delivered, and how *accurate* (error-free) it is



• Figure 5: Collaborative Planning Usage Patterns

[REF-07]. There is thus a need to be able to specify and manage a broad range of QoS properties, more than the typical set of multimedia-based properties, which focus on communication/bandwidth properties and have a real-time and LAN-based flavor

4.2.1 Broad Scope of QoS

The need for adaptation raises difficult problems of systems engineering. There are many dimensions over which the usage and environment of applications may vary, for example:

- The type of user task that is to be performed.
- The size of the user group.
- The extent of the network (global or local).
- The capacity and loading factors of the network.

There is no such thing as a universal application, of course; variations in types of user tasks demand the development of entirely new applications for some new sets of requirements. But taken altogether, there are too many variables in the world of collaborative workspaces to code a new application from the ground up for each new combination of requirements. Not only do we want our applications to adapt to changing circumstances, but we want the bulk of code we wrote for one set of circumstances (capturing the function of the application) still to be used in new circumstances.

QoS middleware for collaborative workspaces thus must allow us to combine solutions to these problems, including:

- User interaction (task-specific functionality) at the application level.
- Resource issues and adaptation at the middleware level.
- Transport details at a lower level.

4.2.2 Adaptation to Changes in Quality of Service

Even within a single distributed execution of a single application, there can be wide variations in QoS requirements of collaborative workspace applications. For example:

Some groups of users may be tightly coupled to each other, but only loosely coupled to other individuals or groups.

Some users may be located in next-door offices, others at opposite sides of the globe.

Network conditions may be poor for some users, excellent for others.

These requirements also can change dynamically as time passes. Developers of collaborative applications would like to be able to field a family of applications which are intended for new operating environments, and they require systematic support for QoS to be able to achieve this.

5 Collaborative Workspaces and Quality of Service (QoS) Metrics

As shown in the preceding section, collaborative workspaces will in many cases require some non-trivial level of QoS. Such a requirement, however, is meaningless unless we can specify the level of QoS that is needed or desired by applications and their users. Likewise, we cannot determine whether the requirement is being met unless we have some way to measure the QoS actually obtained by the application. We address these two issues in this section.

5.1 Metrics

In order to tailor its activation the meta-object needs quantification of the service behavior and needs of the collaboration it is supporting. Two principal categories of metrics are those defined from the perspective of a single user and those defined from the perspective of a group of two or more users. The former is not a pre-requisite for the latter but given local metrics there are many possibilities in how one might combine these to form a global metric.

While it is useful to measure how well the *system* is meeting the collective requirements of its users, actually quantifying this based on metrics for isolated users is difficult to do in a way that is both objective and representative. An example of the difficulties that arise is the consideration of a system in which one user has poor network connectivity, while all the others have good connectivity. To measure how well the system supports poor network situations, one might want to weight the local metrics of the modem user more heavily. On the other hand, what objective numerical basis is selected to weight the metric? The poor performance may be poor "on average", "at peak", or "in variation", and still other parameters of what is in actuality a distribution over time. Which parameters are more important ultimately depend on which aspects of the system you are really trying to measure, and no single choice appears definitive. What kind of aggregation function to use and if aggregation is meaningful will vary from metric to metric.

5.2 Dependability

We will also require second order metrics measuring how well the system meets the constraints requested by its users. The simplest form of these would simply be the total amount of time for any particular constraint fails to be met. Similar aggregation issues exist for this type of metric if the constraints in question are locally defined.

5.3 Mutual Consistency Metrics

This subsection defines one family of metrics for mutual consistency. The idea is counting the number of updates any particular participant has yet to receive. The globalization of this metric is achieved by simply summing the individual update counts. A related metric is the amount of data that needs to be delivered to each party. This weights each update count by the size of the update it refers to. Finally one can weight these sizes by the time it would take to transfer them to each party. Because of varying network conditions, this final metric is an estimate rather than an exact quantity.

5.3.1 Accuracy Metric

An upper bound on the number of network events needed to restore consistency is given by the total number of out-of-date versions, summed over all hosts:

$$\alpha(T) = \sum_i (T - H_i)$$

Notes:

1. $H_i <= T$, so $\alpha(T) >= 0$, and equals 0 only at consistency

2. This is an upper bound on the number of network events needed to achieve zero inconsistency.

3. This is not a completely accurate estimate. This is an over-estimate because it assumes only one update event per network event and no known objects above the greatest lower bound, H. But there is also a small correction in the opposite direction since some duplicate transmissions can happen. This latter correction can be eliminated if one defines a network event to include all re-transmissions necessary.

5.3.2 Precision Metric

There are two related inconsistency metrics for precision. The summation restriction is the same, but the terms of the sum differ.

$$\pi(T) = \sum_i \pi_i(T)$$

$\pi_i(T) = \Sigma U_{kv}$ over all k, v such that $T_{kv} > H_i$

The amount of data which needs to be transferred to achieve $\alpha=0$ is the sum of the individual encodings of state transitions.

5.3.3 Timeliness Metric

The amount of time it would take to transfer this data strongly depends on both transmission strategy and on source distribution. For fully symmetric sources of state transitions and point-to-point transmission:

$$\tau(T) = \sum_i \pi_i(T) M^{-1} \sum_j bandwidth_{ij}^{-1}$$

(the last factors are just the mean inverse bandwidth out of i).

5.3.4 General Notes on Metrics

The α, π, and τ concepts captured by the above metrics should be related measures with similar sum and weight structures. As conceived here, the three different qualities are almost different "units" for consistency, rather than radically distinct measurements.

The skew counting approach detailed above is not the only possible measure. Another possibility would be a generalization to information theoretic entropy measure. This is given by $\Sigma_i p_i log\ p_i$, where the p_i are the probabilities of system state i. Mutual inconsistency presents a few choices on the state space indexed by i here, but the most natural one is the states which the system could be in given the relaxed consistency constraint. If each possible skew state is equally likely, $p_i = 1/N$, and the entropy is in fact simply $log\ N$. If the N in question is the number of possible states the data can be in, this is the product of all the individual possibilities for variation – i.e. the log of the product of 2 raised to the power of the number of bits to represent the discrepancies. The logarithm of that product is thus a sum almost identical to the above proposed inconsistency metric, π(T). Hence entropy metrics are really a generalization of the skew-counting measure, where more information about the p_i is taken into account. This information may come from either an *a priori* model of access patterns or a heuristic probability distribution based upon past history. This entropy metric for inconsistency would lead to another family of derived metrics with data and transfer time scale. Indeed there may even be reasonable physical analogies to entropy-related quantities such as temperature.

5.3.5 Metrics Measurement Issues

Both the simple "skew counting" and the "state counting" measures require near-perfect global information to evaluate. In general any measure of distributed inconsistency will have this problem because even determining if perfect consistency exists requires global data. This unfortunately makes these metrics very expensive to dynamically collect, since it requires synchronization points at every point of measurement.

Determining the behavior of these measured quantities over physical-time relative to algorithms and protocols employed for ensuring consistency can be achieved from a synthesis of locally logged data.

At the end of some significant period of data mutation these local logs can be merged and metrics computed. Even assuming each program is keeping local log information, the question of how much and of what types remains. For example, if there is no need for $\pi(T)$, the size of updates does need not be recorded, or if there is no need for $\tau(T)$ then the physical time at message transmission and receipt need not be saved. It is also important to consider the cost of such detailed logging. Significant performance impact could be incurred.

There are also significant performance impacts in network load incurred by sending the entire series of state information to each process in a session. The burden of continual data collection, redistribution, merging and computation of the metric is likely to be fairly significant, and may negate most gains one might acquire by any adaptive strategy based on the information.

This raises the important practical question of whether there exists a reasonable approximation strategy requiring less than perfect information. Since the system aims to be dynamically adaptive to network conditions.

6 DSW Experiments

For experimental purposes, we constructed a highly simplified example of an application whose quality of service needed to be controlled. In our example application, the server simply returned a large array of 32-bit integers. (Since the contents of the array were not interesting, the array elements were generated "on the fly" by a simple function whenever the server was called.)

The implementation of adaptation consisted of an interface that allowed the application client to specify a range of array elements to be returned by a single CORBA call, specified by the starting position in the array and the number of elements to be returned. This enabled two client algorithms to be investigated:

1. The client could request the entire array in one CORBA call by passing a zero (to position the server at the first element of the array) and the full length of the array. This algorithm is similar to the aggressive prefetching strategy described above for OpenMap.
2. The client could request the contents of the array in N approximately equal pieces by making N CORBA calls, each time passing a different initial index and a length sufficient to fetch all elements up to the next initial index. (The length was approximately 1/N times the full length of the array each time.) This algorithm had characteristics similar to the "chunking" and "streaming" algorithms proposed for OpenMap.

We implemented a client and a server in Java (using JDK 1.1.5) and ran each on a separate, unloaded Pentium PC (one for the client, one for the server) on the Linux operating system. All inter-process communication occurred via the Visibroker for Java 3.0 ORB.

We ran two series of experiments to compare the performance tradeoffs on a LAN with those on a WAN. For the LAN case, we connected the two PCs with a 10 Mb/s Ethernet isolated from the traffic on other machines. This should be considered a near-optimum measurement for the given software running on a network of this type. For the WAN case, we connected the two PCs with a satellite-delay simulation box, set at a clock rate of 62.5 kHz (to simulate as closely as we could a 56k line) and a latency of 32 ms (simulating a radio transmission over a total distance of approximately 10,000 km).

The results are summarized in the following table. In each case we tried various access strategies, including prefetching the entire array at once (represented as 1 subdivision in the table), and various numbers of subdivisions from 2 to 1000. We tried each strategy 21 times, recording the system time before and after each attempt and the difference in seconds between those two times. Thus each possible case was represented by a uniform sample of 21 measurements. Each measurement in the LAN environment transferred 200,000 integers, or about 800KB. In the WAN environment it was necessary to "adapt" the example by sending a smaller array of 10,000 integers (about 40KB) in order for the experiments to be completed in a timely fashion.
Table 3 displays the smallest elapsed time observed for each strategy, the greatest time, and the median time. Here, there were 10 measurements in each sample that were faster than the median time and 10 that were slower; the median time can be regarded as a good measure of the "average" time taken. (The arithmetic mean, in contrast, is subject to being unrepresentative of the sample, because

transient "hiccups" in the operating system could add significant amounts of time to some measurements and skew the sample.)

• Table 3: Experimental Performance on a LAN

Number of subdivisions	Minimum elapsed	Median elapsed	Maximum elapsed
1	3.785	3.909	4.055
2	3.615	3.792	3.858
3	3.569	3.828	4.043
4	3.667	4.079	4.269
6	3.22	3.348	3.686
8	2.269	2.333	2.408
10	2.092	2.147	2.234
15	1.782	1.793	1.916
20	1.701	1.721	1.781
25	1.708	1.735	1.797
30	1.733	1.75	1.786
40	1.675	1.688	1.707
50	1.692	1.71	1.741
60	1.732	1.744	1.816
70	1.678	1.688	1.787
80	1.694	1.701	1.732
90	1.728	1.731	1.785
100	1.742	1.762	1.819
200	2.103	2.151	2.224
1000	4.296	5.331	6.425

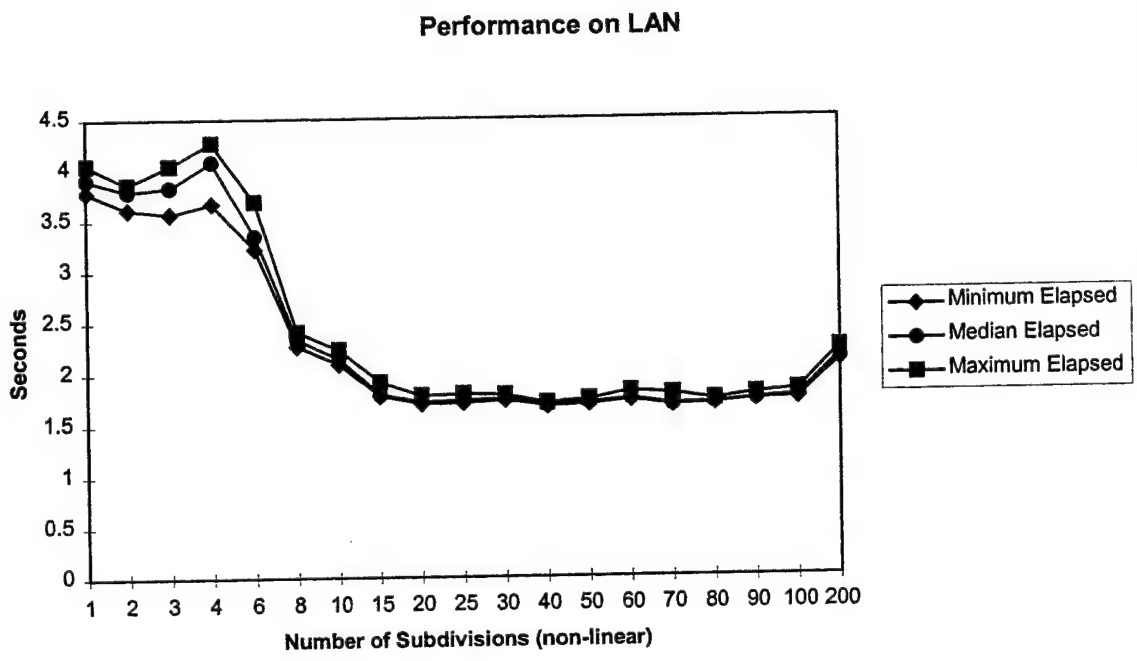
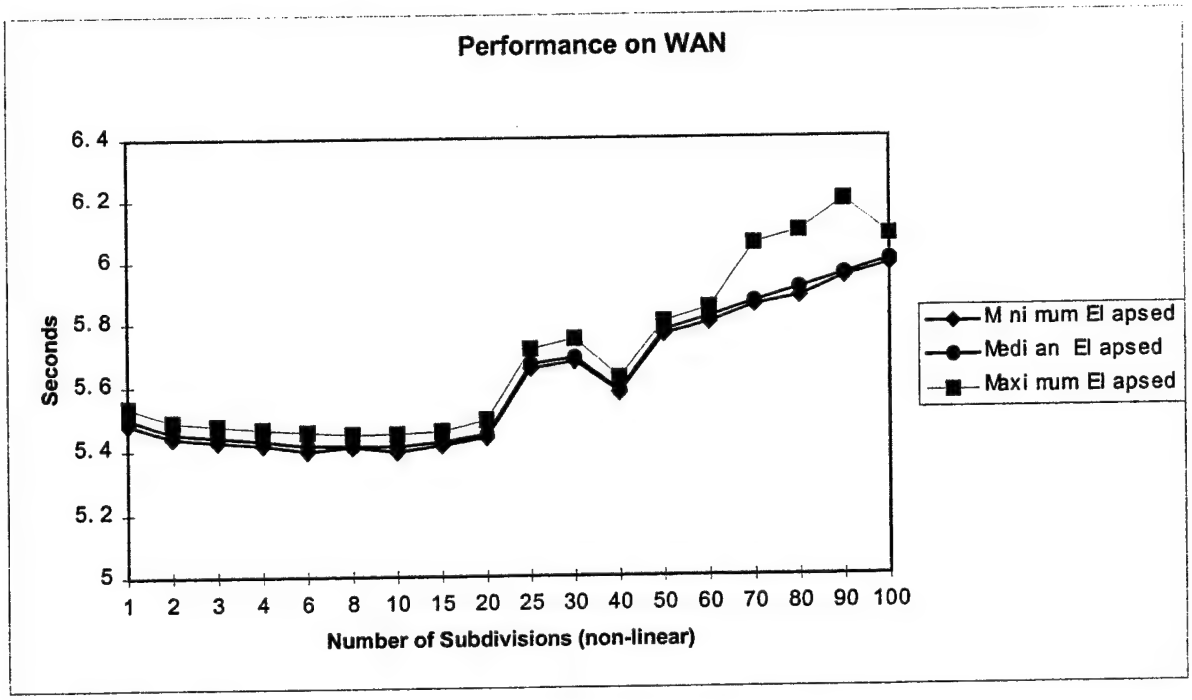


Figure 6 : Elapsed Time in seconds vs. subdivisions of the returned data on a LAN

• Table 4: Experimental Performance on a WAN

Number of subdivisions	Minimum elapsed	Median elapsed	Maximum elapsed
1	5.483	5.501	5.535
2	5.439	5.453	5.489
3	5.427	5.441	5.477
4	5.417	5.43	5.465
6	5.397	5.414	5.456
8	5.41	5.414	5.449
10	5.395	5.412	5.45
15	5.415	5.425	5.458
20	5.438	5.448	5.492
25	5.653	5.668	5.715
30	5.676	5.688	5.748
40	5.577	5.582	5.623
50	5.762	5.774	5.802
60	5.799	5.818	5.847
70	5.855	5.866	6.056
80	5.882	5.91	6.096
90	5.946	5.955	6.197
100	5.986	6.001	6.082
200	6.748	6.889	6.935
1000	12.591	14.944	24.631



• Figure 7: Elapsed time in seconds vs. subdivisions of returned data, on a simulated WAN

These observations are explained by the following facts:

1. The costs for marshaling and unmarshaling the CORBA request were trivial. In contrast, the CPU costs for marshaling and unmarshaling the reply were large (on the order of seconds). Half of this cost was incurred at the server and half at the client. When the entire array was fetched at once, these costs had to be incurred in sequence; but when the array was fetched in pieces, the client was able to unmarshal one piece while the server marshaled the next. Hence the subdivided requests benefited from concurrency, and might be expected to run in less elapsed time.
2. Each individual CORBA call incurred a certain amount of overhead. The greater the number of subdivisions of the array, the greater the total overhead incurred by that strategy. When the array was subdivided into a sufficiently large number of pieces, the added cost of this overhead overwhelmed the speedup attributable to the added concurrency, and the elapsed time increased. (The measurements for 1000 subdivisions were included to illustrate how dramatic the slowdown could be.)
3. In the WAN environment, the costs tended to be dominated by the long latencies and low bandwidth of the communications channel, rather than by the CPU cost of marshaling and unmarshaling data. (In contrast, marshaling costs were far higher in the LAN in comparison to network costs.) The overhead associated with each CORBA message is greater in a low-bandwidth environment, and so the concurrency gains due to subdivision of the array

represented less of an end-to-end speedup factor than was observed in the LAN, and disappeared at a much lower degree of subdivision.

We found that the optimal elapsed times for this application in a LAN environment occurred for subdivisions in the range between 15 and 100 (with local minima at about 40 and 70 subdivisions), with a speedup by more than a factor of 2, whereas the optimal elapsed times for the WAN occurred at approximately 10 subdivisions with a speedup of only about 2 percent. Strategies that were optimal for the LAN case (e.g., 70 subdivisions) actually slowed overall response times in the WAN.

It should be noted that in order to obtain even the observed performance in the WAN environment, it was necessary to reduce the amount of data sent. Hence two adaptations really were being applied separately. The question might be raised whether the observed differences were due merely to the adaptation in the amount of data sent. Indeed, experiments returning 40KB arrays on the LAN showed very little benefit in subdividing the data, similar to the observations on the simulated WAN. An alternative explanation is that there is an optimal "frame length" for the returned data; for example, if it is optimal to divide a 40KB array in 10 pieces, then one should simply divide any array into pieces of length 4KB. We note, however, that while it was better to divide the 40KB array into pieces of 4KB each than pieces of 20KB each, the opposite was true of the 800KB array. So whether our property is "number of pieces" or "size of each piece," it must still vary in order to adapt to different environments and requirements.

7 Conclusions and Future Work

Collaborative workspaces represent a complex but increasing important type of application. These applications must adapt to changes in system resources, the environment, or user requirements in order to maintain an acceptable quality of service. Adaptation will become increasingly important as collaborative applications become more widespread and are applied in more and more non-trivial environments, such as wide-area networks and environments that are under attack. This is especially important when the functions for which the collaborative applications are desired (for example, crisis management) are also the most likely to threaten disruption of the communications network or other resources.

Quality-of-service can be categorized as constraints on timeliness, precision, or accuracy of the information transmitted among applications, or among different parts of an application. There is increasing evidence for the need to measure quality of service, as well as increased understanding of these metrics.

Applications and demonstrations developed at BBN Technologies illustrate the need for adaptation to meet quality-of-service goals, as shown in sections 3 and 4 of this report as well as in separate deliverables produced at BBN. The QuO framework is a promising approach to addressing this problem.

Much work remains to be done to develop quality-of-service facilities for collaborative workspaces that can be used as standard practice. Work continues at BBN technologies to integrate practical quality-of-service tools and concepts into the QuO framework, and to build a toolkit to make this functionality available to application programmers.

References

[REF-01] Carnegie Mellon University, METOC Anchor Desk Public Page. Internet Publication, http://mako.evol.ri.cmu.edu/wxad/home.html.

[REF-02] GTE Internetworking. Distributed Spatial Technology Laboratory. Internet Publication, http://javamap.bbn.com/.

[REF-03] GTE Internetworking. OpenMap(tm) Map System Reference Guide. Internet Publication, http://javamap.bbn.com/matt/.

[REF-04] GTE Internetworking. Logistics Anchor Desk. Internet Publication, http://openmap.bbn.com/lad/.

[REF-05] GTE Internetworking. VALAD — Voice Activated Logistics Anchor Desk. Internet Publication, http://openmap.bbn.com/lad/Future-Technology/VALAD/VALAD.html.

[REF-06] Karr, D.A., Bakken, D.E., Zinky, J.A., and Lawrence, T.F. Towards Quality of Service for Groupware, submitted to *19th International Conference on Distributed Computing Systems (ICDCS'99)*, IEEE, Austin, Texas, May 31 to June 4, 1999.

[REF-07] Sabata, B., Chatterjee, S., Davis, M., Sydir, J.J., and Lawrence, T. Taxonomy for QoS Specifications. In *Proceedings of The Third Workshop on Object-Oriented Real-Time Dependable Systems (WORDS '97)*, IEEE, February 1997.

MISSION
OF
AFRL/INFORMATION DIRECTORATE (IF)

The advancement and application of information systems science and technology for aerospace command and control and its transition to air, space, and ground systems to meet customer needs in the areas of Global Awareness, Dynamic Planning and Execution, and Global Information Exchange is the focus of this AFRL organization. The directorate's areas of investigation include a broad spectrum of information and fusion, communication, collaborative environment and modeling and simulation, defensive information warfare, and intelligent information systems technologies.